**Modelo de Domínio Anêmico**

Este é um daqueles antipadrões que existe há muito tempo, mas parece estar tendo um surto específico no momento. Eu estava conversando com Eric Evans sobre isso, e nós dois percebemos que eles parecem estar ficando mais populares. Como grandes impulsionadores de um Modelo de Domínio adequado, isso não é bom.

O sintoma básico de um modelo de domínio anêmico é que, à primeira vista, ele se parece com a coisa real. Existem objetos, muitos com o nome de substantivos no espaço de domínio, e esses objetos estão conectados com os relacionamentos e a estrutura ricos que os verdadeiros modelos de domínio têm. O problema surge quando você olha para o comportamento e percebe que dificilmente há qualquer comportamento nesses objetos, tornando-os pouco mais do que pacotes de getters e setters. Na verdade, muitas vezes esses modelos vêm com regras de design que dizem que você não deve colocar nenhuma lógica de domínio nos objetos de domínio. Em vez disso, há um conjunto de objetos de serviço que capturam toda a lógica do domínio, realizando todos os cálculos e atualizando os objetos do modelo com os resultados. Esses serviços estão no topo do modelo de domínio e usam o modelo de domínio para dados.

O horror fundamental desse antipadrão é que ele é muito contrário à ideia básica do design orientado a objetos; que é combinar dados e processos juntos. O modelo de domínio anêmico é na verdade apenas um projeto de estilo procedimental, exatamente o tipo de coisa que fanáticos por objeto como eu (e Eric) têm lutado desde nossos primeiros dias em Smalltalk. O que é pior, muitas pessoas pensam que objetos anêmicos são objetos reais e, portanto, perdem completamente o objetivo do design orientado a objetos.

Bem, o purismo orientado a objetos está muito bem, mas eu percebo que preciso de argumentos mais fundamentais contra essa anemia. Em essência, o problema com modelos de domínio anêmicos é que eles incorrem em todos os custos de um modelo de domínio, sem produzir nenhum dos benefícios. O custo principal é a dificuldade de mapear para um banco de dados, que normalmente resulta em uma camada inteira de mapeamento O / R. Isso vale a pena se você usar as poderosas técnicas OO para organizar lógicas complexas. Ao puxar todo o comportamento para os serviços, no entanto, você basicamente acaba com os scripts de transação e, portanto, perde as vantagens que o modelo de domínio pode trazer. Conforme discutido em P do EAA, os Modelos de Domínio nem sempre são a melhor ferramenta.

Também vale a pena enfatizar que colocar o comportamento nos objetos de domínio não deve contradizer a abordagem sólida de usar camadas para separar a lógica de domínio de coisas como persistência e responsabilidades de apresentação. A lógica que deve estar em um objeto de domínio é a lógica de domínio - validações, cálculos, regras de negócios - como você quiser chamá-la. (Há casos em que você defende colocar a fonte de dados ou a lógica de apresentação em um objeto de domínio, mas isso é ortogonal à minha visão de anemia.)

Uma fonte de confusão em tudo isso é que muitos especialistas OO recomendam colocar uma camada de serviços procedurais em cima de um modelo de domínio, para formar uma Camada de Serviço. Mas este não é um argumento para tornar o modelo de domínio vazio de comportamento; na verdade, os defensores da camada de serviço usam uma camada de serviço em conjunto com um modelo de domínio rico em comportamento.

O excelente livro de Eric Evans, Domain Driven Design, tem o seguinte a dizer sobre essas camadas.

Camada de aplicação [seu nome para Camada de serviço]: define as tarefas que o software deve fazer e direciona os objetos de domínio expressivos para resolver os problemas. As tarefas pelas quais essa camada é responsável são significativas para os negócios ou necessárias para a interação com as camadas de aplicativos de outros sistemas. Esta camada é mantida fina. Ele não contém regras de negócios ou conhecimento, mas apenas coordena tarefas e delega trabalho para colaborações de objetos de domínio na próxima camada abaixo. Ele não tem um estado que reflete a situação do negócio, mas pode ter um estado que reflete o progresso de uma tarefa para o usuário ou programa. Camada de Domínio (ou Camada de Modelo): Responsável por representar conceitos do negócio, informações sobre a situação do negócio e regras de negócio. O estado que reflete a situação do negócio é controlado e usado aqui, embora os detalhes técnicos de armazenamento sejam delegados à infraestrutura. Essa camada é o coração do software empresarial.

O ponto chave aqui é que a camada de serviço é fina - toda a lógica principal está na camada de domínio. Ele reitera este ponto em seu padrão de serviço:

Agora, o erro mais comum é desistir muito facilmente de ajustar o comportamento em um objeto apropriado, deslizando gradualmente para a programação procedural.

Não sei por que esse antipadrão é tão comum. Suspeito que seja devido a muitas pessoas que realmente não trabalharam com um modelo de domínio adequado, especialmente se eles vêm de um histórico de dados. Algumas tecnologias o encorajam; como Entity Beans do J2EE, que é uma das razões pelas quais prefiro modelos de domínio POJO. Em geral, quanto mais comportamento você encontrar nos serviços, maior será a probabilidade de se privar dos benefícios de um modelo de domínio. Se toda a sua lógica está nos serviços, você se roubou às cegas.